```python

\_date'), State('date-range', 'end\_date'), State('model-selection', 'value')]

)

def update\_combined\_predictions(n\_clicks, stock\_symbol, start\_date, end\_date, selected\_models):

# Fetch data from MySQL based on inputs

predicted\_data = fetch\_predictions\_from\_mysql(mysql\_connection\_string)

# Filter data by date range

filtered\_data = predicted\_data.loc[start\_date:end\_date]

# Create traces for the actual close prices and predictions from selected models

traces = [go.Scatter(x=filtered\_data.index, y=filtered\_data['Close'], mode='lines', name='Actual Close')]

for model in selected\_models:

traces.append(go.Scatter(x=filtered\_data.index, y=filtered\_data[model + '\_Pred'], mode='lines', name=model + ' Prediction'))

# Create the figure

figure = {

'data': traces,

'layout': {

'title': 'Combined Predictions vs Actual Close Price',

'xaxis': {'title': 'Date'},

'yaxis': {'title': 'Price'}

}

}

return figure

@app.callback(

Output('technical-analysis', 'figure'),

[Input('submit-button', 'n\_clicks')],

[State('stock-symbol', 'value'), State('date-range', 'start\_date'), State('date-range', 'end\_date')]

)

def update\_technical\_analysis(n\_clicks, stock\_symbol, start\_date, end\_date):

# Fetch data from MySQL based on inputs

predicted\_data = fetch\_predictions\_from\_mysql(mysql\_connection\_string)

# Filter data by date range

filtered\_data = predicted\_data.loc[start\_date:end\_date]

# Create traces for technical indicators

traces = [

go.Scatter(x=filtered\_data.index, y=filtered\_data['Close'], mode='lines', name='Close'),

go.Scatter(x=filtered\_data.index, y=filtered\_data['SMA\_20'], name='SMA 20'),

go.Scatter(x=filtered\_data.index, y=filtered\_data['SMA\_50'], name='SMA 50'),

go.Scatter(x=filtered\_data.index, y=filtered\_data['EMA\_50'], name='EMA 50')

]

# Create the figure

figure = {

'data': traces,

'layout': {

'title': 'Technical Analysis',

'xaxis': {'title': 'Date'},

'yaxis': {'title': 'Price'}

}

}

return figure

@app.callback(

Output('volume-plot', 'figure'),

[Input('submit-button', 'n\_clicks')],

[State('stock-symbol', 'value'), State('date-range', 'start\_date'), State('date-range', 'end\_date')]

)

def update\_volume\_plot(n\_clicks, stock\_symbol, start\_date, end\_date):

# Fetch data from MySQL based on inputs

predicted\_data = fetch\_predictions\_from\_mysql(mysql\_connection\_string)

# Filter data by date range

filtered\_data = predicted\_data.loc[start\_date:end\_date]

# Create the volume plot

figure = {

'data': [go.Bar(x=filtered\_data.index, y=filtered\_data['Volume'], name='Volume')],

'layout': {

'title': 'Trade Volume',

'xaxis': {'title': 'Date'},

'yaxis': {'title': 'Volume'}

}

}

return figure

@app.callback(

Output('rsi-plot', 'figure'),

[Input('submit-button', 'n\_clicks')],

[State('stock-symbol', 'value'), State('date-range', 'start\_date'), State('date-range', 'end\_date')]

)

def update\_rsi\_plot(n\_clicks, stock\_symbol, start\_date, end\_date):

# Fetch data from MySQL based on inputs

predicted\_data = fetch\_predictions\_from\_mysql(mysql\_connection\_string)

# Filter data by date range

filtered\_data = predicted\_data.loc[start\_date:end\_date]

# Create the RSI plot

figure = {

'data': [go.Scatter(x=filtered\_data.index, y=filtered\_data['RSI'], mode='lines', name='RSI')],

'layout': {

'title': 'Relative Strength Index (RSI)',

'xaxis': {'title': 'Date'},

'yaxis': {'title': 'RSI'}

}

}

return figure

@app.callback(

Output('macd-plot', 'figure'),

[Input('submit-button', 'n\_clicks')],

[State('stock-symbol', 'value'), State('date-range', 'start\_date'), State('date-range', 'end\_date')]

)

def update\_macd\_plot(n\_clicks, stock\_symbol, start\_date, end\_date):

# Fetch data from MySQL based on inputs

predicted\_data = fetch\_predictions\_from\_mysql(mysql\_connection\_string)

# Filter data by date range

filtered\_data = predicted\_data.loc[start\_date:end\_date]

# Create the MACD plot

figure = {

'data': [go.Scatter(x=filtered\_data.index, y=filtered\_data['MACD'], mode='lines', name='MACD')],

'layout': {

'title': 'Moving Average Convergence Divergence (MACD)',

'xaxis': {'title': 'Date'},

'yaxis': {'title': 'MACD'}

}

}

return figure

@app.callback(

Output('atr-plot', 'figure'),

[Input('submit-button', 'n\_clicks')],

[State('stock-symbol', 'value'), State('date-range', 'start\_date'), State('date-range', 'end\_date')]

)

def update\_atr\_plot(n\_clicks, stock\_symbol, start\_date, end\_date):

# Fetch data from MySQL based on inputs

predicted\_data = fetch\_predictions\_from\_mysql(mysql\_connection\_string)

# Filter data by date range

filtered\_data = predicted\_data.loc[start\_date:end\_date]

# Create the ATR plot

figure = {

'data': [go.Scatter(x=filtered\_data.index, y=filtered\_data['ATR'], mode='lines', name='ATR')],

'layout': {

'title': 'Average True Range (ATR)',

'xaxis': {'title': 'Date'},

'yaxis': {'title': 'ATR'}

}

}

return figure

@app.callback(

Output('obv-plot', 'figure'),

[Input('submit-button', 'n\_clicks')],

[State('stock-symbol', 'value'), State('date-range', 'start\_date'), State('date-range', 'end\_date')]

)

def update\_obv\_plot(n\_clicks, stock\_symbol, start\_date, end\_date):

# Fetch data from MySQL based on inputs

predicted\_data = fetch\_predictions\_from\_mysql(mysql\_connection\_string)

# Filter data by date range

filtered\_data = predicted\_data.loc[start\_date:end\_date]

# Create the OBV plot

figure = {

'data': [go.Scatter(x=filtered\_data.index, y=filtered\_data['OBV'], mode='lines', name='OBV')],

'layout': {

'title': 'On-Balance Volume (OBV)',

'xaxis': {'title': 'Date'},

'yaxis': {'title': 'OBV'}

}

}

return figure

@app.callback(

Output('trading-signals', 'figure'),

[Input('submit-button', 'n\_clicks')],

[State('stock-symbol', 'value'), State('date-range', 'start\_date'), State('date-range', 'end\_date')]

)

def update\_trading\_signals(n\_clicks, stock\_symbol, start\_date, end\_date):

# Fetch data from MySQL based on inputs

predicted\_data = fetch\_predictions\_from\_mysql(mysql\_connection\_string)

# Filter data by date range

filtered\_data = predicted\_data.loc[start\_date:end\_date]

# Create the trading signals plot

figure = {

'data': [

go.Scatter(x=filtered\_data.index, y=filtered\_data['Close'], name='Close'),

go.Scatter(x=filtered\_data.index, y=filtered\_data['Signal'], name='Trading Signal', yaxis='y2')

],

'layout': {

'title': 'Trading Signals',

'xaxis': {'title': 'Date'},

'yaxis': {'title': 'Price'},

'yaxis2': {

'title': 'Signal',

'overlaying': 'y',

'side': 'right'

}

}

}

return figure

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server(debug=True)

```

### Steps to Run the Dash App

1. \*\*Set Up the MySQL Database\*\*:

- Ensure MySQL is installed and running on your machine.

- Create a new database and import the initial SQL dump.

```sh

mysql -u your\_username -p

CREATE DATABASE stock\_data\_db;

USE stock\_data\_db;

SOURCE path\_to/stock\_prediction\_project/data/mysql\_database.sql;

```

2. \*\*Install Python Dependencies\*\*:

- Create a virtual environment and install the required dependencies.

```sh

python3 -m venv venv

source venv/bin/activate # On Windows use `venv\Scripts\activate`

pip install -r requirements.txt

```

3. \*\*Train and Save Machine Learning Models\*\*:

- Execute the script to train the models and save the results into MySQL.

```sh

python models/ml\_models.py

```

4. \*\*Run the Dash Application\*\*:

- Execute the Dash app script.

Sure! Here is a proposed directory structure for your project and detailed steps on how to run the Dash app.

### Directory Structure

```

stock\_prediction\_project/

│

├── data/

│ └── mysql\_database.sql # Initial SQL dump for setting up the MySQL database

│

├── models/

│ └── ml\_models.py # Script for training and saving ML models

│

├── utils/

│ ├── fetch\_data.py # Script to fetch data from MySQL

│ ├── technical\_analysis.py # Script for computing technical indicators

│ ├── trading\_signals.py # Script for generating trading signals

│ └── db\_utils.py # Script for interacting with the database

│

├── app/

│ └── dash\_app.py # Main Dash application

│

├── requirements.txt # Python dependencies

│

└── README.md # Project documentation

```

### Step-by-Step Instructions to Run the Dash App

1. \*\*Set Up the MySQL Database\*\*:

- Ensure MySQL is installed and running on your machine.

- Create a new database and import the initial SQL dump.

```sh

mysql -u your\_username -p

CREATE DATABASE stock\_data\_db;

USE stock\_data\_db;

SOURCE path\_to/stock\_prediction\_project/data/mysql\_database.sql;

```

2. \*\*Install Python Dependencies\*\*:

- Create a virtual environment and install the required dependencies.

```sh

python3 -m venv venv

source venv/bin/activate # On Windows use `venv\Scripts\activate`

pip install -r requirements.txt

```

3. \*\*Train and Save Machine Learning Models\*\*:

- Execute the script to train the models and save the results into MySQL.

```sh

python models/ml\_models.py

```

4. \*\*Run the Dash Application\*\*:

- Execute the Dash app script.

```sh

python app/dash\_app.py

```

### Detailed Script Files

#### utils/fetch\_data.py

```python

import pandas as pd

from sqlalchemy import create\_engine

def fetch\_data\_from\_mysql(mysql\_connection\_string):

engine = create\_engine(mysql\_connection\_string)

query = "SELECT Date, Open, High, Low, Close, Volume FROM stock\_data"

df = pd.read\_sql(query, con=engine)

df['Date'] = pd.to\_datetime(df['Date'])

df.set\_index('Date', inplace=True)

return df

```

#### utils/technical\_analysis.py

```python

import ta

def compute\_technical\_indicators(df):

df['SMA\_20'] = ta.trend.sma\_indicator(df['Close'], window=20)

df['SMA\_50'] = ta.trend.sma\_indicator(df['Close'], window=50)

df['EMA\_50'] = ta.trend.ema\_indicator(df['Close'], window=50)

df['RSI'] = ta.momentum.rsi(df['Close'], window=14)

df['MACD'] = ta.trend.macd\_diff(df['Close'])

df['ATR'] = ta.volatility.average\_true\_range(df['High'], df['Low'], df['Close'], window=14)

df['OBV'] = ta.volume.on\_balance\_volume(df['Close'], df['Volume'])

return df

```

#### utils/trading\_signals.py

```python

def generate\_trading\_signals(df):

df['Signal'] = 0

df['Signal'][df['SMA\_20'] > df['SMA\_50']] = 1

df['Signal'][df['SMA\_20'] < df['SMA\_50']] = -1

df['Position'] = df['Signal'].diff()

return df

```

#### utils/db\_utils.py

```python

import pandas as pd

from sqlalchemy import create\_engine

def store\_results\_to\_mysql(df, predictions, mysql\_connection\_string):

engine = create\_engine(mysql\_connection\_string)

result\_df = df[['Close', 'SMA\_20', 'SMA\_50', 'EMA\_50', 'RSI', 'MACD', 'ATR', 'OBV', 'Signal', 'Position']].iloc[-len(predictions['LinearRegression']):].copy()

for model\_name, pred in predictions.items():

result\_df[model\_name + '\_Pred'] = pred

result\_df.to\_sql('stock\_predictions', con=engine, if\_exists='replace')

def fetch\_predictions\_from\_mysql(mysql\_connection\_string):

engine = create\_engine(mysql\_connection\_string)

query = "SELECT \* FROM stock\_predictions"

df = pd.read\_sql(query, con=engine)

df['Date'] = pd.to\_datetime(df['Date'])

df.set\_index('Date', inplace=True)

return df

```

#### models/ml\_models.py

```python

import pandas as pd

from sqlalchemy import create\_engine

from sklearn.linear\_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor

from sklearn.svm import SVR

from sklearn.neural\_network import MLPRegressor

from xgboost import XGBRegressor

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from utils.fetch\_data import fetch\_data\_from\_mysql

from utils.technical\_analysis import compute\_technical\_indicators

from utils.trading\_signals import generate\_trading\_signals

from utils.db\_utils import store\_results\_to\_mysql

mysql\_connection\_string = 'mysql+pymysql://username:password@localhost/stock\_data\_db'

stock\_data = fetch\_data\_from\_mysql(mysql\_connection\_string)

stock\_data = compute\_technical\_indicators(stock\_data)

stock\_data = generate\_trading\_signals(stock\_data)

def preprocess\_data(df):

df['Return'] = df['Close'].pct\_change()

df.dropna(inplace=True)

features = ['Open', 'High', 'Low', 'Volume', 'Return']

target = 'Close'

X = df[features]

y = df[target]

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, shuffle=False)

return X\_train, X\_test, y\_train, y\_test, scaler

X\_train, X\_test, y\_train, y\_test, scaler = preprocess\_data(stock\_data)

models = {

'LinearRegression': LinearRegression(),

'RandomForest': RandomForestRegressor(n\_estimators=100),

'SVR': SVR(),

'MLPRegressor': MLPRegressor(hidden\_layer\_sizes=(100,), max\_iter=500),

'XGBoost': XGBRegressor(n\_estimators=100)

}

predictions = {}

for model\_name, model in models.items():

model.fit(X\_train, y\_train)

predictions[model\_name] = model.predict(X\_test)

store\_results\_to\_mysql(stock\_data, predictions, mysql\_connection\_string)

```

#### app/dash\_app.py

```python

import dash

import dash\_core\_components as dcc

import dash\_html\_components as html

import plotly.graph\_objs as go

from dash.dependencies import Input, Output, State

from utils.db\_utils import fetch\_predictions\_from\_mysql

mysql\_connection\_string = 'mysql+pymysql://username:password@localhost/stock\_data\_db'

predicted\_data = fetch\_predictions\_from\_mysql(mysql\_connection\_string)

app = dash.Dash(\_\_name\_\_)

app.layout = html.Div([

html.H1("Stock Price Prediction Dashboard"),

dcc.Tabs(id="tabs", children=[

dcc.Tab(label='Price Prediction', children=[

html.Div([

html.Label("Select Stock Symbol:"),

dcc.Input(id='stock-symbol', value='AAPL', type='text'),

html.Label("Select Date Range:"),

dcc.DatePickerRange(

id='date-range',

start\_date=predicted\_data.index.min(),

end\_date=predicted\_data.index.max()

),

html.Label("Select Models:"),

dcc.Checklist(

id='model-selection',

options=[

{'label': 'Linear Regression', 'value': 'LinearRegression'},

{'label': 'Random Forest', 'value': 'RandomForest'},

{'label': 'SVR', 'value': 'SVR'},

{'label': 'MLP Regressor', 'value': 'MLPRegressor'},

{'label': 'XGBoost', 'value': 'XGBoost'}

],

value=['LinearRegression', 'RandomForest', 'SVR', 'MLPRegressor', 'XGBoost']

),

html.Button(id='submit-button', n\_clicks=0, children='Submit'),

dcc.Graph(id='combined-predictions')

])

]),

dcc.Tab(label='Technical Analysis', children=[

html.Div([

dcc.Graph(id='technical-analysis')

])

]),

dcc.Tab(label='Volume Analysis', children=[

html.Div([

dcc.Graph(id='volume-plot')

])

]),

dcc.Tab(label='Performance Metrics', children=[

html.Div([

dcc.Graph(id='rsi-plot'),

dcc.Graph(id='macd-plot'),

dcc.Graph(id='atr-plot'),

dcc.Graph(id='obv-plot')

])

]),

dcc.Tab(label='Trading Signals', children=[

html.Div([

dcc.Graph(id='trading-signals')

])

])

])

])

@app.callback(

Output('combined-predictions', 'figure'),

[Input('submit-button', 'n\_clicks')],

[State('stock-symbol', 'value'), State('date-range', 'start